

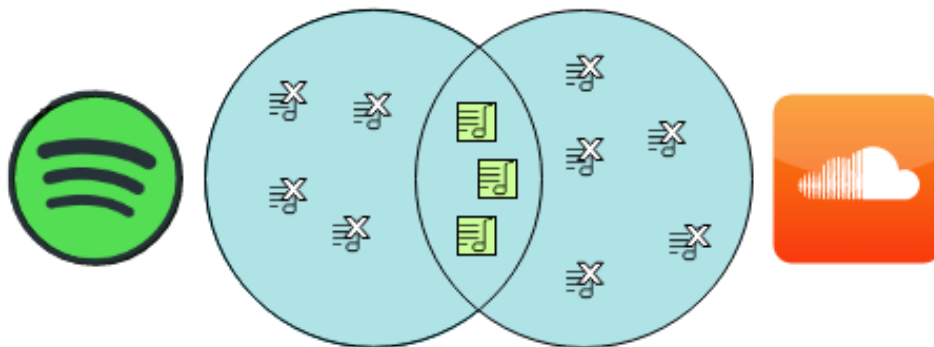
1 2



9 0

FACULDADE DE  
CIÊNCIAS E TECNOLOGIA  
UNIVERSIDADE DE  
COIMBRA

# Secure Multiparty Computation: PSI protocols Assignment #2



Mestrado em Segurança Informática  
Ano letivo 2022/2023

Inês Martins Marçal

Nº: 2019215917

João Carlos Borges Silva

Nº: 2019216753

## Índice

<b>Introdução.....</b>	<b>2</b>
<b>Caracterização do Dataset.....</b>	<b>2</b>
Descrição e objetivo.....	2
Tratamento de colunas.....	2
Atributos.....	3
<b>Aplicação dos protocolos PSI.....</b>	<b>3</b>
Protocolos PSI.....	3
Divisão do dataset.....	5
Naive Hashing.....	6
Diffie-Hellman-based.....	7
OT-based.....	8
Solução implementada.....	9
<b>Benchmark dos protocolos PSI.....</b>	<b>12</b>
Tempo de execução.....	13
Dados trocados entre entidades.....	14
Registos intersetados entre entidades.....	15
Nível de segurança e privacidade.....	16
<b>Sistema de recomendação.....</b>	<b>17</b>
<b>Conclusão.....</b>	<b>17</b>
<b>Referências.....</b>	<b>18</b>

## Introdução

Este trabalho foi realizado no âmbito da cadeira de Segurança e Privacidade com o objetivo de efetuar uma análise detalhada da aplicação de *secure multiparty computation* num *dataset*. Aplicando este método, mais especificamente protocolos de *Private Set Intersection (PSI)*, espera-se que seja possível resolver um problema, definido posteriormente, assim como efetuar o *benchmark* das soluções encontradas.

## Caracterização do Dataset

### Descrição e objetivo

O *dataset* escolhido para a realização deste trabalho consiste num conjunto de *ratings* de músicas classificados por um utilizador em plataformas como o *Spotify* ou o *Soundcloud* (imaginando que estas possibilitam atribuir *ratings* a músicas). Neste caso, serão apresentadas 6 colunas relativamente a cada música: cantor, nome, duração, género, ano de lançamento e *rating* dado pelo utilizador.

Os dados encontram-se organizados da seguinte maneira:

```
Britney Spears,Oops!...I Did It Again,211,R&B,2000,2
blink-182,All The Small Things,167,Blues,1999,4
Faith Hill,Breathe,250,Rock,1999,3
Bon Jovi,It's My Life,224,R&B,2000,2
*NSYNC,Bye Bye Bye,200,Classic,2000,5
Sisqo,Thong Song,253,Classic,1999,3
```

Este *dataset* resulta da utilização de um *dataset* já existente de músicas [1], onde foram apenas extraídos os dados necessários para este trabalho. O significado de cada coluna e transformações efetuadas ao *dataset* original serão especificados nos capítulos seguintes.

Como objetivo principal pretende-se que cada plataforma (*Spotify* e *Soundcloud*) consiga trocar entre si os *ratings* de um utilizador, de forma privada, com o intuito de tornar possível obter a interseção dos dois conjuntos de dados. Através desta interseção, cada plataforma terá conhecimento das músicas que o utilizador fez questão de providenciar um *rating* e com base nisto efetuar sistemas de recomendação de músicas.

### Tratamento de colunas

Considerando que o *dataset* inicial não possuía o mesmo objetivo que o estabelecido para o trabalho em questão, o mesmo teve de sofrer algumas alterações, como a adição, modificação ou remoção de colunas.

O *dataset* original continha 18 colunas (como, por exemplo, cantor, nome da música, duração da música em milissegundos, ano de lançamento da música, entre outras), contudo nem todas as colunas irão ser necessárias para cumprir o objetivo definido. Como tal, foram extraídas as colunas mais importantes como o cantor, nome da música, duração, género de música e ano de lançamento, deixando de parte as restantes.

As colunas de duração e género de música apresentam valores não muito favoráveis para o objetivo descrito e, portanto, foram modificadas de modo a tornar este processo mais simples. A duração da música era apresentada em milissegundos, sendo esta uma escala bastante esparsa, foi alterada para ser apresentada em segundos. Já a coluna do género

musical poderia apresentar mais do que um valor, por exemplo, uma música poderia ser simultaneamente do tipo *Pop* e *Rock*. Assim, esta foi também simplificada, de modo a que cada música apenas possua um gênero associado, sendo atribuído um valor *random* ao nível deste campo no sentido de facilitar este processo. O objetivo é construir um modelo de recomendação e, como tal, os valores dos atributos não terão influência na sua construção.

Finalmente, foi adicionada uma coluna do *rating* atribuído por um utilizador a uma determinada música, sendo esta gerada aleatoriamente. Esta coluna será o foco principal dos modelos de recomendação a serem criados.

### Atributos

Após uma breve apresentação do *dataset* é necessário explicitar melhor o significado dos mesmos, para que não haja quaisquer dúvidas no futuro durante o desenvolvimento de modelos preditivos que poderão ter como base o mesmo. De seguida, encontram-se detalhados os diferentes atributos que compõem *dataset*:

- cantor: artista que produziu a música. Será um atributo importante, já que indivíduos que tenham tendência a ouvir um determinado cantor, terão uma maior probabilidade de o voltar a ouvir.
- música: nome da música
- duração da música: o nome é bem explícito, esta é a coluna que representa o tempo que uma música demora a ser reproduzida, expresso em segundos.
- gênero da música: indica o tipo de música, poderá permitir, eventualmente, a recomendação de músicas do mesmo tipo. Este é composto por diversas categorias: "Jazz", "Rock", "Hip Hop", "Blues", "Classic", "Eletronic", "Pop", "Kpop", "Jpop", "Country", "R&B", "Funk", "Reggae", "Alternative", "Folk".
- ano de lançamento da música: novamente, o nome já é autoexplicativo, significando o ano em que a música foi publicada.
- rating atribuído: classificação que o utilizador atribui à música, sendo o fator principal a considerar durante a criação de modelos de recomendação. O *rating* poderá apresentar valores de 1 a 5.

Estes serão, assim, os atributos que representam os dados em questão e os quais se pretende trocar/interseção de forma segura e privada, utilizando os protocolos *PSI*.

## Aplicação dos protocolos PSI

### Protocolos PSI

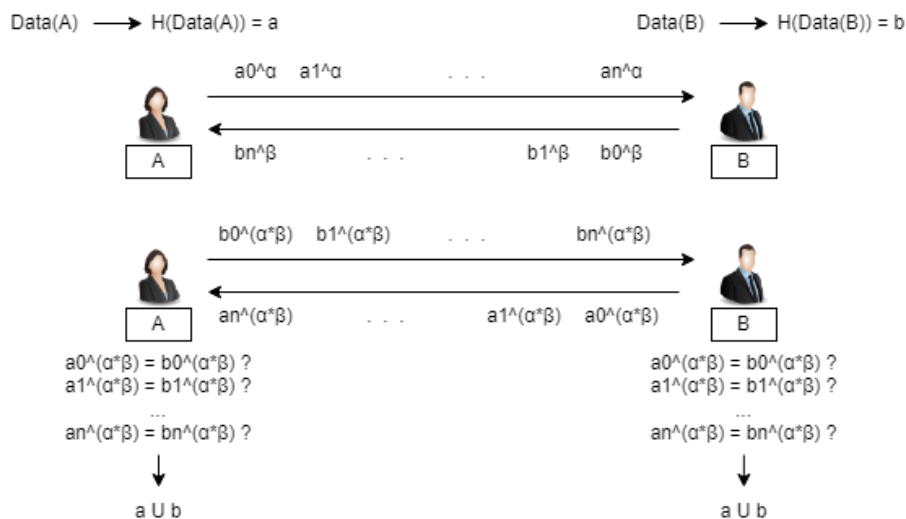
Os protocolos *Private Set Intersection (PSI)* constituem sucintamente mecanismos que permitem a 2 ou mais entidades descobrir a interseção dos seus dados, por exemplo, possibilitam que 2 indivíduos compreendam os contactos que têm em comum. Este processo deverá ser efetuado de forma segura e privada, isto é, nenhum dos lados consegue descobrir os dados enviados pelo outro, apenas perceber quais destes estão em comum entre si.

Estes protocolos apresentam diversas implementações como:

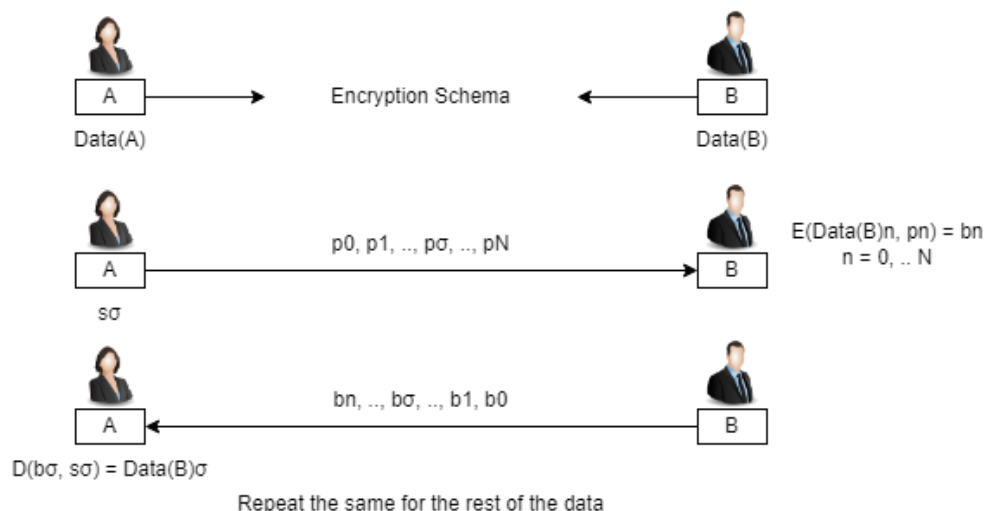
- Naive hashing: esta primeira abordagem é bastante simples. Considerando apenas 2 entidades (A e B), ambas chegam a um consenso de usar a função de *hash*  $H(x)$ . Posto isto, B envia os seus dados encriptados com a função  $H(x)$ , A compara os seus *hashes* com os enviados por B e reenvia a interseção entre estes, mais especificamente os que apresentam valores idênticos.



- Diffie-Hellman-based: este protocolo acaba por ser bastante semelhante ao anterior. Neste caso, é utilizado ainda o algoritmo de *diffie-hellman*, ou seja, cada *hash* enviado será elevado a um determinado expoente (A eleva a *alfa* e B eleva a *beta*) e, posteriormente, ao ser recebido é elevado, novamente, a outro expoente (B eleva a *beta* e A eleva a *alfa*). Desta forma, cada lado da comunicação terá todos os *hashes* (de ambos os lados) elevados a *alfa* e *beta*, aos quais irá proceder à sua interseção.



- OT-based: esta abordagem aplica o mesmo procedimento observado até agora, mas com a diferença de ser utilizado *OT (oblivious-transfer)*<sup>1</sup>. Este é um tipo de protocolo que permite a troca de informação entre 2 (ou mais) entidades, tal que as mesmas somente fiquem a conhecer a interseção entre os dois conjuntos de dados.



<sup>1</sup> Por exemplo: Imaginando que A pretende saber uma das 4 mensagens de B. A gera um par de mensagens público-privadas e 3 chaves públicas. Estas chaves públicas são enviadas para B, encriptando as suas mensagens com as mesmas. B envia as mensagens encriptadas para A. Como A apenas possui uma chave privada que corresponde a uma das chaves públicas enviadas, logo apenas poderá descriptar uma mensagem e aprender a mesma. Deste modo, como B desconhece a chave privada de A não saberá qual foi a mensagem que esta conseguiu descriptar.

- Solução nova: esta constitui a solução que irá ser implementada com vista a resolver o problema inicial, ou seja, descobrir a interseção entre 2 *datasets* de forma segura e privada. A mesma será especificada em capítulos mais adiante.

### Divisão do dataset

Com o intuito de efetuar a avaliação e *benchmarking* das diferentes implementações, o *dataset* original teve de ser dividido em 2 partes (uma para o *Spotify* e outra para o *Soundcloud*). Esta divisão deve ser feita de modo a haver uma conjunto de dados semelhantes entre as duas partes e, conseqüentemente, tornar possível a sua interseção. Este processo deverá ser repetido pelo menos 5 vezes, onde cada parte da comunicação obterá um número crescente de dados, isto é, ambas as entidades começam por ter inicialmente 10 músicas, depois aumenta para 100, 1000, etc.

Para os primeiros testes será utilizada uma versão de um *dataset* de músicas relativamente pequeno, com 2066 músicas. Este deverá ser dividido em 2 partes, de modo a ser feita uma primeira abordagem aos diferentes protocolos. A utilização deste primeiro dataset é de grande importância para a elaboração da solução proposta, que visa, precisamente, resolver o problema de *PSI (Privacy-Preserving Set Intersection)*.

A divisão do *dataset* será feita de forma *random*, mas ao mesmo tempo controlada, de modo a que, como referido, existam realmente elementos das duas partes do dataset que sejam iguais e, por sua vez, venham a permitir a sua interseção. De seguida, é explicitado o algoritmo (código) que possibilita efetuar esta mesma divisão:

```
def gen_data(n):
    size = 2066-1
    if n > 1033:
        print("Error: use a <n> lower than 1033")

    n_intersets = random.randint(math.floor(n*0.3), math.floor(n*0.7))

    data = []
    part1 = []; part2 = []
```

Inicialmente, é recebido um parâmetro 'n' que determina a quantidade de dados atribuída a cada parte. É verificado se este valor é maior que metade do *dataset*, de modo a evitar quaisquer erros e são criados dois grupos para guardar os dados de ambas as partes. É de notar que o número de interseções irá variar entre 30% a 70% do valor "n", visto considerar-se esta a percentagem mais adequada.

```
with open("new_new.csv", "r") as f:
    data = copy.deepcopy(f.readlines())
    for _ in range(n):
        if n_intersets > 0:
            index = random.randint(0, size)
            part1.append(data[index]); part2.append(data[index])
            data.remove(data[index])
            n_intersets -= 1
            size -= 1
        else:
            index = random.randint(0, size)
            part1.append(data[index])
            data.remove(data[index])
            size -= 1
```

```

index = random.randint(0, size)

part2.append(data[index])

data.remove(data[index])

size -= 1

```

Acima encontra-se demonstrada a leitura do ficheiro “new\_new.csv”, *dataset* inicial com apenas 2066 músicas. De seguida, são percorridas as linhas deste ficheiro onde, de forma *random*, são escolhidas algumas das músicas para serem inseridas em ambas as partes (*part1* e *part2*), constituindo a interseção dos 2 conjuntos. O restante espaço de cada entidade é preenchido por músicas *random* diferentes.

```

random.shuffle(part1)

random.shuffle(part2)

```

```

with open("part1.csv", "w") as f2:

    f2.write("".join(part1))

```

```

with open("part2.csv", "w") as f3:

    f3.write("".join(part2))

```

Por fim, é aplicado um pequeno *shuffle* às músicas de cada conjunto, a fim de evitar que as músicas da interseção se encontrem todas seguidas. O conteúdo de cada parte é escrito em diferentes ficheiros, que serão usados, como já referido, para efetuar um primeiro estudo das diferentes abordagens PSI.

São solicitadas 5 formas diferentes de dividir o *dataset*, cada uma com tamanhos distintos, mais especificamente: 100, 1000, 10000, 100000, 1000000. Acredita-se que estas serão as divisões mais adequadas para avaliar o desempenho dos diferentes protocolos *PSI*.

## Naive Hashing

Como referido, este método é bastante simples, onde para verificar a interseção entre dois conjuntos, duas entidades enviam os *hashes* dos seus dados e comparam no sentido de compreender se existe algum tipo de interseção entre os mesmos. Tal como já mencionado, irão realizar-se testes com *datasets* de diferentes tamanhos. Primeiramente, efetua-se um teste com o *dataset* apresentado na alínea anterior, dividindo o mesmo em duas partes de 1033:

Capturing from lo						Details			
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help						File			
Apply a display filter ... <Ctrl-/>						Name: /tmp/wireshark_loX3LA51.pcapng			
						Length: 18 kb			
						Hash (SHA256): 63217363697eef88ca3c7ca3f657327c1043ee0106fc952e619f135fb3fda0			
						Hash (RIPEMD160): e8090909024ee5841428bc076c50448b681			
						Hash (SHA1): 08664232e8a3221b74428664b0e0c553020m5			
						Format: Wireshark/... pcapng			
						Encapsulation: Ethernet			
						Time			
						First packet: 2023-05-23 12:39:12			
						Last packet: 2023-05-23 12:39:12			
						Elapsed: 00:00:00			
						Capture			
						Hardware: Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz (with SSE4.2)			
						OS: Linux 5.19.0-41-generic			
						Application: Dumpcap (Wireshark) 3.6.2 (GK v3.6.2 packaged as 3.6.2-2)			
						Interfaces			
						Interface lo			
						Dropped packets: Unknown			
						Capture filter: none			
						Link type: Ethernet			
						Packet size limit (bytes): 65535 bytes			
						Statistics			
						Measurement			
						Packets: 22 (100.0%)			
						Time span: 0.038			
						Average pps: 571.7			
						Average packet size, B: 796			
						Bytes: 17508 (100.0%)			
						Average bytes/s: 454 k			
						Average bits/s: 3639 k			
						Capture file comments			
No.	Time	Source	Destination	Protocol	Length	Info			
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	41986 → 7766 [SYN] Seq=1			
2	0.000015276	127.0.0.1	127.0.0.1	TCP	74	7766 → 41986 [SYN, ACK]			
3	0.000027839	127.0.0.1	127.0.0.1	TCP	66	41986 → 7766 [ACK] Seq=1			
4	0.000049948	127.0.0.1	127.0.0.1	TCP	70	7766 → 41986 [PSH, ACK]			
5	0.000059904	127.0.0.1	127.0.0.1	TCP	66	41986 → 7766 [ACK] Seq=1			
6	0.000074622	127.0.0.1	127.0.0.1	TCP	82	7766 → 41986 [PSH, ACK]			
7	0.000077413	127.0.0.1	127.0.0.1	TCP	66	41986 → 7766 [ACK] Seq=1			
8	0.01147607	127.0.0.1	127.0.0.1	TCP	70	41986 → 7766 [PSH, ACK]			
9	0.011491305	127.0.0.1	127.0.0.1	TCP	66	7766 → 41986 [ACK] Seq=2			
10	0.011521315	127.0.0.1	127.0.0.1	TCP	70	41986 → 7766 [PSH, ACK]			
11	0.011525186	127.0.0.1	127.0.0.1	TCP	66	7766 → 41986 [ACK] Seq=2			
12	0.011535565	127.0.0.1	127.0.0.1	TCP	70	41986 → 7766 [PSH, ACK]			
13	0.011539947	127.0.0.1	127.0.0.1	TCP	66	7766 → 41986 [ACK] Seq=2			
14	0.011547942	127.0.0.1	127.0.0.1	TCP	70	41986 → 7766 [PSH, ACK]			
15	0.011550866	127.0.0.1	127.0.0.1	TCP	66	7766 → 41986 [ACK] Seq=2			
16	0.011559800	127.0.0.1	127.0.0.1	TCP	70	41986 → 7766 [PSH, ACK]			
17	0.011562690	127.0.0.1	127.0.0.1	TCP	66	7766 → 41986 [ACK] Seq=2			
18	0.015284643	127.0.0.1	127.0.0.1	TCP	8866	7766 → 41986 [PSH, ACK]			
19	0.018264739	127.0.0.1	127.0.0.1	TCP	8866	41986 → 7766 [PSH, ACK]			
20	0.036891987	127.0.0.1	127.0.0.1	TCP	66	7766 → 41986 [FIN, ACK]			
21	0.039471137	127.0.0.1	127.0.0.1	TCP	66	41986 → 7766 [FIN, ACK]			
22	0.039484247	127.0.0.1	127.0.0.1	TCP	66	7766 → 41986 [ACK] Seq=8			

Como se pode observar pelas imagens são transmitidos 17508 Bytes durante a execução deste protocolo. Sabendo que foram encontradas 371 interseções entre os dois



*datasets* utilizados, algo que seria de esperar devido ao tamanho dos dados trocados, já que os ficheiros originais têm à volta de 47095 *Bytes*. Fazendo um cálculo de quanto espaço seria ocupado por linha (47095 *Bytes* a dividir por 1033 linhas), daria por volta de 45 *Bytes*, multiplicando este valor por 371 (interseções), contabiliza-se cerca de 16914 *Bytes*. Os restantes *Bytes* considerados pelo *wireshark* corresponderão, provavelmente, ao estabelecimento da comunicação ou a outros mecanismos que o protocolo TCP utilize durante a manutenção da mesma.

```
Computation finished. Found 371 intersecting elements:
Take That,Patience,202,Hip Hop,2006,4
Becky G,Sin Pijama,188,Jazz,2018,3
Chingy,One Call Away,276,Eletronic,2003,1
Demi Lovato,Confident,205,Kpop,2015,4
Nina Sky,Move Ya Body,232,Kpop,2004,5
```

Outro fator a ter em conta na avaliação das diferentes abordagens de protocolos *PSI* é o tempo que os mesmos demoram a efetuar as suas operações, já que é importante existir um bom balanço entre eficiência e memória despendida. Através da informação fornecida pelo *Wireshark*, foi possível observar que este processo (interseção de conjuntos de dados), utilizando a abordagem “Naive Hashing”, demorou 0.038 segundos. Este acaba por ser um tempo relativamente pequeno, pelo que é demonstrada a eficiência do mesmo.

## Diffie-Hellman-based

Nesta abordagem, acabou por ter-se algo muito parecido com o que foi visto anteriormente, sendo adicionado ainda o algoritmo de *Diffie-Hellman*. Espera-se, naturalmente, que este protocolo seja um pouco mais pesado em relação ao *Naive Hashing*, já que, como é possível ver em capítulos anteriores, são trocados muitos mais dados entre as duas entidades da comunicação. Utilizando, novamente, o *dataset* de testes, obtiveram-se os seguintes resultados através do *Wireshark* :

The screenshot displays the Wireshark network protocol analyzer. The main pane shows a list of captured packets, with packet 1 selected. The details pane on the right provides information about the selected packet, including file name, length, hashes, and capture details.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	53596 → 7766 [SYN] Seq=0 Win=0
2	0.000014878	127.0.0.1	127.0.0.1	TCP	74	7766 → 53596 [SYN, ACK] Seq=0
3	0.000027482	127.0.0.1	127.0.0.1	TCP	66	53596 → 7766 [ACK] Seq=1 Ack=
4	0.000032652	127.0.0.1	127.0.0.1	TCP	74	7766 → 53596 [ACK] Seq=1 Ack=
5	0.000735588	127.0.0.1	127.0.0.1	TCP	66	53596 → 7766 [ACK] Seq=1 Ack=
6	0.000744834	127.0.0.1	127.0.0.1	TCP	70	7766 → 53596 [PSH, ACK] Seq=5
7	0.000748313	127.0.0.1	127.0.0.1	TCP	66	53596 → 7766 [ACK] Seq=1 Ack=
8	0.000753438	127.0.0.1	127.0.0.1	TCP	70	7766 → 53596 [PSH, ACK] Seq=5
9	0.000755511	127.0.0.1	127.0.0.1	TCP	66	53596 → 7766 [ACK] Seq=1 Ack=
10	0.000758395	127.0.0.1	127.0.0.1	TCP	70	7766 → 53596 [PSH, ACK] Seq=5
11	0.000761317	127.0.0.1	127.0.0.1	TCP	66	53596 → 7766 [ACK] Seq=1 Ack=
12	0.000765135	127.0.0.1	127.0.0.1	TCP	70	7766 → 53596 [PSH, ACK] Seq=5
13	0.000767737	127.0.0.1	127.0.0.1	TCP	66	53596 → 7766 [ACK] Seq=1 Ack=
14	0.011058831	127.0.0.1	127.0.0.1	TCP	70	53596 → 7766 [PSH, ACK] Seq=5
15	0.011148215	127.0.0.1	127.0.0.1	TCP	66	7766 → 53596 [ACK] Seq=21 Ack=
16	0.011156714	127.0.0.1	127.0.0.1	TCP	70	53596 → 7766 [PSH, ACK] Seq=5
17	0.011171992	127.0.0.1	127.0.0.1	TCP	66	7766 → 53596 [ACK] Seq=21 Ack=
18	0.011177008	127.0.0.1	127.0.0.1	TCP	70	53596 → 7766 [PSH, ACK] Seq=5
19	0.011180094	127.0.0.1	127.0.0.1	TCP	66	7766 → 53596 [ACK] Seq=21 Ack=
20	0.011184115	127.0.0.1	127.0.0.1	TCP	70	53596 → 7766 [PSH, ACK] Seq=5
21	0.011186101	127.0.0.1	127.0.0.1	TCP	66	7766 → 53596 [ACK] Seq=21 Ack=
22	0.011190238	127.0.0.1	127.0.0.1	TCP	70	53596 → 7766 [PSH, ACK] Seq=5
23	0.011192321	127.0.0.1	127.0.0.1	TCP	66	7766 → 53596 [ACK] Seq=21 Ack=
24	0.430195476	127.0.0.1	127.0.0.1	TCP	32834	7766 → 53596 [ACK] Seq=21 Ack=
25	0.430200986	127.0.0.1	127.0.0.1	TCP	4298	7766 → 53596 [PSH, ACK] Seq=3
26	0.433799653	127.0.0.1	127.0.0.1	TCP	66	53596 → 7766 [ACK] Seq=21 Ack=
27	0.433964079	127.0.0.1	127.0.0.1	TCP	32834	53596 → 7766 [ACK] Seq=21 Ack=
28	0.433971391	127.0.0.1	127.0.0.1	TCP	66	7766 → 53596 [ACK] Seq=37921
29	0.433977654	127.0.0.1	127.0.0.1	TCP	4298	53596 → 7766 [PSH, ACK] Seq=3
30	0.433979879	127.0.0.1	127.0.0.1	TCP	66	7766 → 53596 [ACK] Seq=37921

**Details**

**File**

- Name: /tmp/wireshark\_ioA9GBS1.pcapng
- Length: 109 kB
- Hash (SHA256): ec3d607dea1d3bccc9245283318327093c3abb8c79af94a49e4a1a29c4708fe
- Hash (RIPEMD160): d0d94e2737942d180ad3f4904b35985628dab04
- Hash (SHA1): c91e131d96c3b08b620d482996a6166f27f85e
- Format: Wireshark - pcapng
- Encapsulation: Ethernet

**Time**

- First packet: 2023-05-24 12:16:41
- Last packet: 2023-05-24 12:16:42
- Elapsed: 00:00:00

**Capture**

- Hardware: Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz (with SSE4.2)
- OS: Linux 5.19.0-41-generic
- Application: Dumpcap (Wireshark) 3.6.2 (Git v3.6.2 packaged as 3.6.2-2)

**Interfaces**

Interface	Dropped packets	Capture filter	Link type	Packet size limit (snaplen)
lo	Unknown	none	Ethernet	262144 bytes

**Statistics**

Measurement	Captured	Displayed	Marked
Packets	34	34 (100.0%)	—
Time span, s	0.921	0.921	—
Average pps	36.9	36.9	—
Average packet size, B	3185	3185	—
Bytes	108300	108300 (100.0%)	—
Average bytes/s	117 k	117 k	—

**Capture file comments**

Desta vez, como se pode observar, a quantidade de dados é bastante maior do que o visto anteriormente, sendo que desta vez foram transmitidos quase 10 vezes mais dados. Este aumento deve-se, principalmente, à forma como o algoritmo funciona, já que, naturalmente, há mais trocas de informação entre duas entidades (primeiro há troca dos *hashes* elevado a um expoente e depois devolução destes mesmos *hashes* elevados a outro expoente). A partir dos pacotes enviados, consegue-se ainda deduzir que muito provavelmente os expoentes utilizados rondam o valor 4. Poderá pensar-se nisto, já que os pacotes de trocas de *hashes* elevados a um expoente são praticamente o quádruplo do tamanho dos ficheiros originais (sabendo que o número de interseções é 371 na mesma).



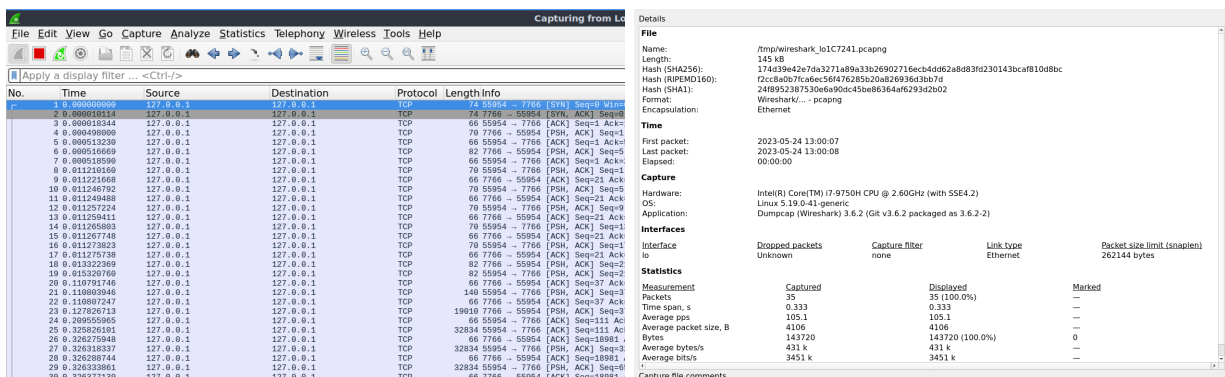
Novamente, os restantes *Bytes* poderão estar a ser utilizados para questões de estabelecimento de comunicação ou até para a implementação de *Diffie-Hellman* utilizada.

```
sti1@sti1:~/SP/Implementation$ ./demo.exe -r 1 -p 2 -f part2.csv
Computation finished. Found 371 intersecting elements:
Enrique Iglesias,Escape,208,Reggae,2001,5
Travis,Sing,228,R&B,2001,1
Blue,U Make Me Wanna - Radio Edit,222,Rock,2003,4
G-Unit,Wanna Get To Know You,265,Kpop,2003,3
Boys Like Girls,Love Drunk,226,Funk,2009,4
```

Atendendo à eficiência da abordagem utilizada pode-se perceber que, neste caso, o tempo aumenta também um pouco relativamente à *Naive Hashing*, atingindo quase 1 segundo. Este volta a ser um comportamento totalmente normal, pelas razões enunciadas durante a análise da memória despendida. Acaba por ser um bom equilíbrio entre eficiência e privacidade, já que com a *Diffie-Hellman* o nível de segurança da comunicação acaba por crescer associado a um custo temporal bastante baixo.

## OT-based

De seguida, avalia-se, numa primeira instância, a abordagem *OT-based*, que volta a ser muito semelhante com o que foi visto nas duas últimas, sendo desta vez usada OT, um tipo de mecanismo explicado anteriormente. É, novamente, esperado que a quantidade de dados e tempo de execução aumentem, devido ao número de trocas de informação efetuadas entre as entidades da comunicação. De seguida, encontram-se apresentados os resultados utilizando o *dataset* de testes criado:



The screenshot shows the Wireshark interface with a packet capture of a TCP connection. The packet list on the left shows 30 packets. The packet details pane on the right shows the selected packet (No. 1) with its structure: Ethernet II, Internet Protocol Version 4, and Transmission Control Protocol. The packet statistics pane on the right shows the overall statistics for the capture, including the number of packets, bytes, and the distribution of packet sizes.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.0.0.1	172.0.0.1	TCP	74	55954 → 7766 [SYN] Seq=0 Win=0
2	0.00001014	172.0.0.1	172.0.0.1	TCP	74	7766 → 55954 [ACK] Seq=1
3	0.000018344	172.0.0.1	172.0.0.1	TCP	68	55954 → 7766 [ACK] Seq=1 Ack=1
4	0.000498809	172.0.0.1	172.0.0.1	TCP	70	7766 → 55954 [PSH, ACK] Seq=1
5	0.000513239	172.0.0.1	172.0.0.1	TCP	68	55954 → 7766 [ACK] Seq=1 Ack=1
6	0.000516669	172.0.0.1	172.0.0.1	TCP	82	7766 → 55954 [PSH, ACK] Seq=5
7	0.000518099	172.0.0.1	172.0.0.1	TCP	68	55954 → 7766 [ACK] Seq=1 Ack=1
8	0.011210169	172.0.0.1	172.0.0.1	TCP	70	55954 → 7766 [PSH, ACK] Seq=1
9	0.01121668	172.0.0.1	172.0.0.1	TCP	68	7766 → 55954 [ACK] Seq=21 Ack=1
10	0.011246702	172.0.0.1	172.0.0.1	TCP	70	55954 → 7766 [PSH, ACK] Seq=5
11	0.011249488	172.0.0.1	172.0.0.1	TCP	68	7766 → 55954 [ACK] Seq=21 Ack=1
12	0.011257224	172.0.0.1	172.0.0.1	TCP	70	55954 → 7766 [PSH, ACK] Seq=9
13	0.011259411	172.0.0.1	172.0.0.1	TCP	68	7766 → 55954 [ACK] Seq=21 Ack=1
14	0.011265983	172.0.0.1	172.0.0.1	TCP	70	55954 → 7766 [PSH, ACK] Seq=1
15	0.011267748	172.0.0.1	172.0.0.1	TCP	68	7766 → 55954 [ACK] Seq=21 Ack=1
16	0.011273823	172.0.0.1	172.0.0.1	TCP	70	55954 → 7766 [PSH, ACK] Seq=1
17	0.011277738	172.0.0.1	172.0.0.1	TCP	68	7766 → 55954 [ACK] Seq=21 Ack=1
18	0.011322369	172.0.0.1	172.0.0.1	TCP	82	7766 → 55954 [PSH, ACK] Seq=2
19	0.01320769	172.0.0.1	172.0.0.1	TCP	82	55954 → 7766 [PSH, ACK] Seq=2
20	0.110791746	172.0.0.1	172.0.0.1	TCP	68	7766 → 55954 [ACK] Seq=37 Ack=1
21	0.110803946	172.0.0.1	172.0.0.1	TCP	148	55954 → 7766 [PSH, ACK] Seq=3
22	0.110807247	172.0.0.1	172.0.0.1	TCP	68	7766 → 55954 [ACK] Seq=37 Ack=1
23	0.12786713	172.0.0.1	172.0.0.1	TCP	19010	7766 → 55954 [PSH, ACK] Seq=3
24	0.269050965	172.0.0.1	172.0.0.1	TCP	68	55954 → 7766 [ACK] Seq=111 Ack=1
25	0.325826181	172.0.0.1	172.0.0.1	TCP	32834	55954 → 7766 [ACK] Seq=111 Ack=1
26	0.326275949	172.0.0.1	172.0.0.1	TCP	68	7766 → 55954 [ACK] Seq=18881 Ack=1
27	0.326318337	172.0.0.1	172.0.0.1	TCP	32834	55954 → 7766 [PSH, ACK] Seq=3
28	0.326288744	172.0.0.1	172.0.0.1	TCP	68	7766 → 55954 [ACK] Seq=18881 Ack=1
29	0.326333861	172.0.0.1	172.0.0.1	TCP	32834	55954 → 7766 [PSH, ACK] Seq=6
30	0.326377139	172.0.0.1	172.0.0.1	TCP	68	7766 → 55954 [ACK] Seq=18881 Ack=1

Como seria de esperar, o número de *Bytes* transmitidos voltou a ser bastante grande, ultrapassando quase o tamanho original 10 vezes. O facto de serem efetuadas diversas trocas de informação, nomeadamente OTs, é o principal fator para este comportamento, já que são realizadas diversas tentativas até que uma das entidades apresente os conteúdos da outra e obtenha a interseção dos dois conjuntos. Inclusive a própria ferramenta demonstra o número OTs efetuadas, tendo, neste caso, sido 1200.

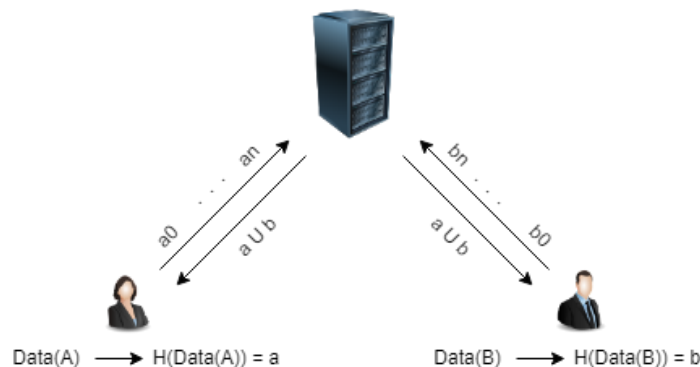
```
sti1@sti1:~/SP/Implementation$ ./demo.exe -r 1 -p 3 -f part2.csv
Hashing 1000 elements with arbitrary length into 8 bytes
Client: bins = 1200, elebitlen = 54 and maskbitlen = 64 and performs 1200 OTs
Computation finished. Found 371 intersecting elements:
Enrique Iglesias,Escape,208,Reggae,2001,5
Travis,Sing,228,R&B,2001,1
Blue,U Make Me Wanna - Radio Edit,222,Rock,2003,4
```

Quanto ao tempo de execução volta novamente a ser bastante baixo, mas atingindo quase 1 segundo. Comparando com *Diffie-Hellman*, o tempo acaba por ser mais baixo, mas volta a ser algo normal se atendermos ao algoritmo de *hashing*/criptação usado na abordagem *OT-based*, enquanto que *Diffie-Hellman* utiliza operações como exponenciação. O segundo processo acaba por ser mais custoso, pelo que é normal uma ligeira diferença na eficiência destas duas abordagens.

Ainda assim, *PSI protocol* com *OT*, acaba por oferecer um bom balanço entre eficiência, memória despendida e privacidade/seurança durante a interseção dos dois conjuntos.

### Solução implementada

Era solicitado no enunciado que fosse implementada uma solução para este tipo de protocolos, utilizando uma *trusted centralized party* (ou seja, um servidor intermediário). É importante considerar que esta implementação pressupõe que o servidor intermediário entre as duas entidades seja confiável e não tente enganar qualquer uma das partes. Esta solução funciona da seguinte forma:



Na prática, é efetuado o seguinte processo: as duas entidades enviam cada um dos seus dados encriptados com algoritmo *hashing*/criptográfico, o servidor calcula a interseção dos dados recebidos e reenvia para cada uma das entidades os *hashes* intercetados. Com isto, tanto A como B conseguem perceber os dados que têm em comum, sem que cada um deles saiba por completo os dados do outro lado.

Quanto ao código da solução, o mesmo funciona de duas maneiras: servidor ou cliente. Nas imagens que se seguem, a primeira representa a execução do código em modo servidor e a segunda em modo cliente:

```
stipl1@stipl1:~/SP/Implementation$ make
g++ -c -o server.o server.cpp
g++ -std=c++11 -Wall server.o client.o psi_server_aided.o ut
stipl1@stipl1:~/SP/Implementation$ ./psi_server_aided -r 0
stipl1@stipl1:~/SP/Implementation$
```

```
stipl1@stipl1:~/SP/Implementation$ ./psi_server_aided -r 1 -ip 192.168.1.121 -f part1.csv
Found 371 intersections
Take That,Patience,202,Hip Hop,2006,4
Becky G,Sin Pijama,188,Jazz,2018,3
Chingy,One Call Away,276,Electronic,2003,1
Demi Lovato,Confident,205,Kpop,2015,4
Nina Sky,Move Ya Body,232,Kpop,2004,5
will.i.am,This Is Love,279,Funk,2013,3
Billie Eilish,Bored,180,Funk,2017,1
Flo Rida,GDFR (feat. Sage the Gemini & Lookas),190,Blues,2015,3
P!nk,Who Knew,208,Reggae,2006,4
```

Enquanto que em modo servidor só é indicada a *role* como argumento (0 - servidor ou 1 cliente), em modo cliente além da *role*, é introduzido o *ip* do servidor ao qual se pretende conectar e o ficheiro de dados a utilizar.

Para iniciar o protocolo, é executado em primeiro lugar o servidor e este aguarda que eventualmente 2 clientes se conectem.

```
if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    perror("socket failed");
    exit(EXIT_FAILURE);
}

if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt, sizeof(opt))) {
    perror("setsockopt");
    exit(EXIT_FAILURE);
}
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(PORT);

if (bind(server_fd, (struct sockaddr*)&address, sizeof(address)) < 0) {
    perror("bind failed");
    exit(EXIT_FAILURE);
}

if (listen(server_fd, 3) < 0) {
    perror("listen");
    exit(EXIT_FAILURE);
}

if ((new_socket[0] = accept(server_fd, (struct sockaddr*)&address, (socklen_t*)&addrlen)) < 0) {
    perror("accept");
    exit(EXIT_FAILURE);
}

if ((new_socket[1] = accept(server_fd, (struct sockaddr*)&address, (socklen_t*)&addrlen)) < 0) {
    perror("accept");
    exit(EXIT_FAILURE);
}
```

Por sua vez, o código do cliente, ao ser executado, começa por ler o ficheiro que contém os respetivos dados e conecta-se ao servidor com o IP especificado

```
file.open(filename, std::ios::in);
if (file){
    while(getline(file, music)){
        data.emplace_back(music);
    }
    file.close();
}
else{
    std::cout << "File doesn't exist\n";
}

. . . . .

if ((status = connect(client_fd, (struct sockaddr*)&serv_addr, sizeof(serv_addr))) < 0) {
    printf("\nConnection Failed \n");
    return 1;
}
```

De seguida, cada cliente envia os seus dados encriptados com *hash (sha256)* para o servidor (2ª imagem), o mesmo efetua a interceção entre os dois conjuntos e devolve esse mesmo resultado (1ª imagem). Finalmente, cada cliente verifica que elementos realmente estão em comum com os dados da outra entidade (2ª imagem).

```

recv(new_socket[0], len[0], MAXCHAR, 0);
for(int i=0; i< std::stoi(len[0]); i++){
    recv(new_socket[0], buffer[0], MAXCHAR, 0);
    send(new_socket[0], "ack", strlen("ack"), 0);
    data1.emplace_back(buffer[0]);
}

recv(new_socket[1], len[1], MAXCHAR, 0);
for(int i=0; i< std::stoi(len[1]); i++){
    recv(new_socket[1], buffer[1], MAXCHAR, 0);
    send(new_socket[1], "ack", strlen("ack"), 0);
    data2.emplace_back(buffer[1]);
}

data = interset(data1, data2);

for(int i=0; i<data.size(); i++){
    send(new_socket[0], data[i].c_str(), MAXCHAR,0);
    recv(new_socket[0], buffer[0], MAXCHAR, 0);
    send(new_socket[1], data[i].c_str(), MAXCHAR,0);
    recv(new_socket[1], buffer[1], MAXCHAR, 0);

    //std::cout << data[i] << "\n";
}

send(client_fd, std::to_string(data.size()).c_str(), MAXCHAR,0);
for(int i=0; i< data.size(); i++){
    send(client_fd, sha256(data[i]).c_str(), MAXCHAR,0);
    recv(client_fd, buffer, MAXCHAR, 0);
}

while(1){
    recv(client_fd, buffer, MAXCHAR, 0);
    if(strcmp(buffer,"ack")==0){
        break;
    }

    send(client_fd, "ack", strlen("ack"), 0);
    hash.emplace_back(buffer);
}

hash = reverse(data, hash);
std::cout << "Found " << hash.size() << " intersections\n";
for(int i=0; i<hash.size(); i++){
    std::cout << hash[i] << "\n";
}

```

Uma pequena diferença, em relação às implementações das abordagens anteriormente observadas, é que na solução criada por nós procede-se ao envio de um dado de cada vez. Embora este processo possa tornar esta etapa um pouco mais ineficiente, o mesmo permitirá minimizar a perda de pacotes na troca dos dados entre as duas entidades.

Após esta primeira apresentação da solução implementada, será novamente efetuada uma análise tanto em termos de dados trocados, como do tempo de execução da mesma. Segue-se o teste efetuado através do *Wireshark*:

The screenshot displays the Wireshark network protocol analyzer interface. The main pane shows a list of 30 captured packets. The selected packet (No. 1) is a TCP Reset (RST) from 192.168.1.121 to 192.168.1.121. The details pane on the right shows the packet structure, including Ethernet II, Internet Protocol Version 4, and Transmission Control Protocol. The packet bytes pane shows the raw data of the packet.

Relativamente à quantidade de dados enviados, observa-se uma grande diferença em relação às abordagens já apresentadas, sendo neste caso enviados 550762 *Bytes* e um total de 5515 pacotes. Novamente, a quantidade de trocas de dados efetuadas é a principal razão por detrás de um volume de dados tão grande, já que cada linha do ficheiro dos clientes é enviada separadamente de outras, refletindo-se num maior *overhead* no fluxo da comunicação. Outro fator que influenciará muito a quantidade de memória trocada entre cada parte será o tamanho das *strings* enviadas. Enquanto que as abordagens anteriores enviavam apenas parte do *hash*, a implementação aqui demonstrada envia o *hash* (com *sha256*) na sua totalidade, provocando uma troca de grandes quantidades de dados.

```
stipl1@stipl1:~/SP/Implementation$ ./psi_server_aided -r 1 -ip 192.168.1.121 -f part2.csv
Found 371 intersections
Take That,Patience,202,Hip Hop,2006,4
Becky G,Sin Pijama,188,Jazz,2018,3
Chingy,One Call Away,276,Electronic,2003,1
Demi Lovato,Confident,205,Kpop,2015,4
Nina Sky,Move Ya Body,232,Kpop,2004,5
will.i.am,This Is Love,279,Funk,2013,3
Billie Eilish,Bored,180,Funk,2017,1
Flo Rida,GDFR (feat. Sage the Gemini & Lookas),190,Blues,2015,3
```

Quanto à eficiência desta implementação, o tempo de execução passa a ser de 2 segundos, o que acaba por não ter um impacto muito significativo. A diferença em relação às abordagens anteriores deve-se, novamente, às razões previamente enunciadas e, muito provavelmente, ao algoritmo de interseção utilizado, o qual apresenta uma complexidade de  $O(N^2)$ . Apesar da ineficiência acaba por conseguir ultrapassar a abordagem de *Diffie-Hellman* em termos de taxa média de transferência *Bytes*/segundo.

No geral, esta solução, embora ofereça a segurança esperada (assumindo que a *trusted party* é confiável), possui alguns problemas a nível de eficiência devido aos algoritmos utilizados (interseção e *sha256*).

## Benchmark dos protocolos PSI

Após uma análise mais informal das abordagens anteriormente enunciadas, irá ser efetuado o *benchmark* destas utilizando, precisamente, a mesma ferramenta para executar o protocolo *PSI* em causa (exceto a solução por nós implementada). Para as abordagens *naive hashing*, *diffie-hellman* e *OT* é acrescentada a *flag -t* para efetuar o *benchmark*:

```
stipl1@stipl1:~/SP/Implementation$ ./demo.exe -r 0 -p 0 -f part1.csv -t
Time for reading elements:      0.14 s
Computation finished. Found 650253 intersecting elements:
Required time: 0.8 s
Data sent:      10000020.0 B
Data received:  10000020.0 B
```

Utilizando o comando acima é possível executar o *benchmark* de cada uma destas abordagens, de modo a ser obtido o tempo de execução e a quantidade de dados recebidos e enviados.

Já a solução implementada irá conter uma *flag* que, quando utilizada, permite determinar as métricas anteriormente referidas. Para tal, executa-se o seguinte comando:

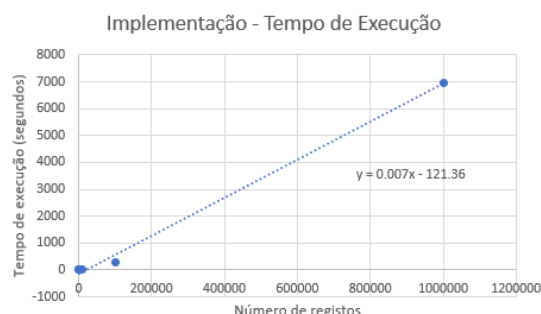
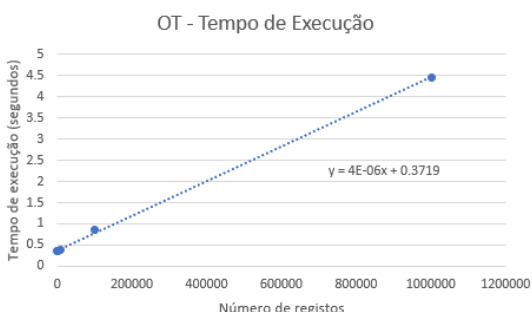
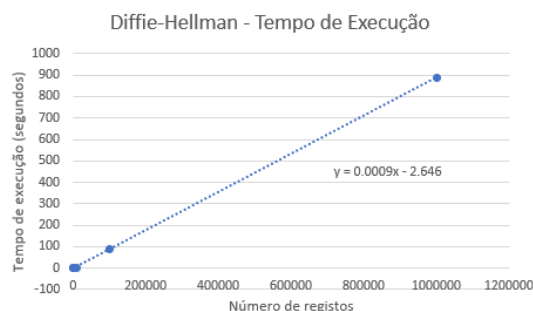
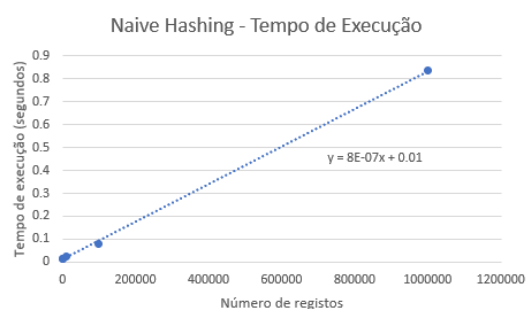
```
stipl1@stipl1:~/SP/Implementation$ ./psi_server_aided -r 0 -t
Required time: 0.269531 s
Data exchanged: 210114 B
```

É de lembrar ainda que durante esta etapa são utilizadas 5 versões de um *dataset* de 2000000 de registos de músicas (gerado artificialmente), que, por sua vez, segue os mesmos moldes do *dataset* demonstrado anteriormente. Estas 5 versões diferem na quantidade de dados trocados entre as duas entidades, mais especificamente: 100, 1000, 10000, 1000000, 1000000.

### Tempo de execução

Trata-se de um ponto já bastante abordado ao longo deste trabalho e no *benchmark* volta a assumir um papel essencial. Avaliar a performance dos protocolos *PSI*, nomeadamente o tempo de execução, é de extrema importância, de modo a entender a sua viabilidade e o seu nível de aptidão para uma determinada situação. Através desta métrica é ainda possível compreender melhor o quão escaláveis os mesmos são e qual o máximo *overhead* atingido durante a execução do processo de *PSI*.

Durante esta etapa são utilizadas as 5 versões de um *dataset*, como referido anteriormente. Seguem-se os resultados obtidos relativamente ao tempo de execução de cada abordagem:



	100	1000	10000	100000	1000000
0	0.016	0.014	0.023	0.077	0.836
2	0.101	0.837	0.827	86.799	890.121
3	0.347	0.345	0.392	0.867	4.459
Our	0.022	0.256	3.037	263.129	6960

Como é possível observar, em todas as abordagens o tempo de execução registado acaba por seguir uma regressão linear. Isto significa que à medida que o tamanho do *dataset* aumenta, o tempo de execução acaba por crescer linearmente. Contudo este comportamento é considerado normal dado a complexidade dos algoritmos usados.

Olhando individualmente para cada uma das abordagens, percebemos que a que apresenta melhores resultados é a *Naive Hashing*, algo de esperar, já que de entre as 4 abordagens é a mais simples e menos custosa (em todos os aspetos). Sem grandes



surpresas, a solução por nós implementada acaba por ser a que revelou piores resultados, uma vez que, diante de um dataset com 1.000.000 de registos, o tempo de execução chegou a quase 3 horas(!). Isto deve-se à falta de otimização dos algoritmos utilizados, os quais, como já referido anteriormente, não são os mais eficientes, como tal os tempos de execução podem tornar-se inaceitáveis para um possível *data mining* entre duas entidades. Caso se atenda aos declives de cada reta, poderá observar-se que quanto pior a solução/abordagem em termos de eficiência, maior será o seu declive.

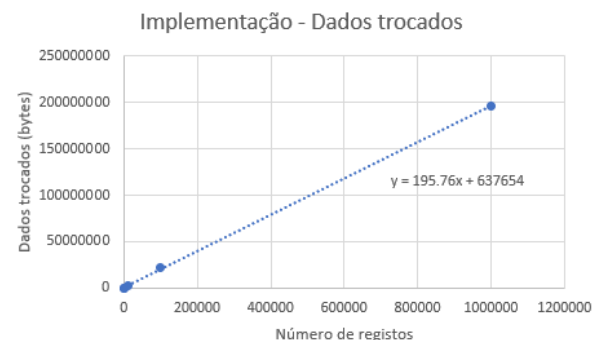
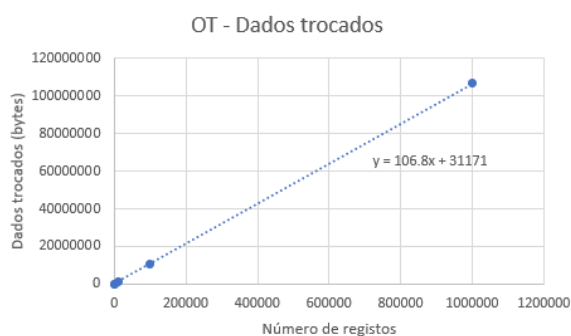
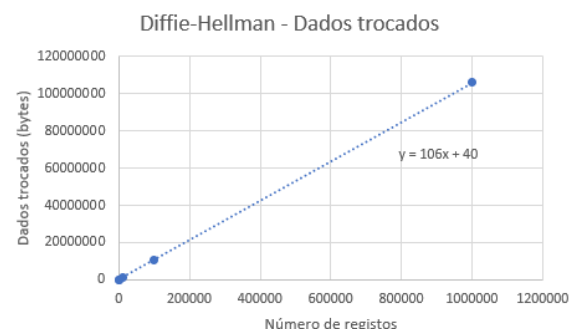
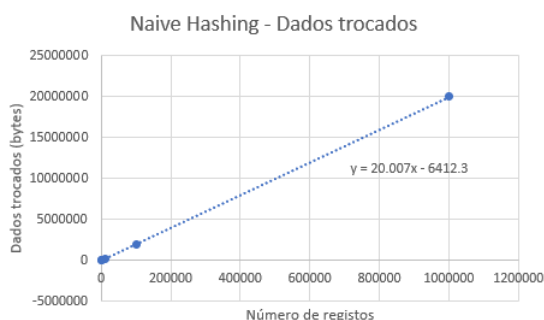
Um dos pontos interessantes em relação aos tempos de execução registados é o facto de, apesar de existir uma maior troca de dados utilizando a abordagem OT, a estratégia que integra *Diffie Hellman* é a que consegue ter uma menor eficiência. Novamente, isto acontece devido ao tipo de operações que ambas utilizam. Outra informação curiosa, obtida pelos gráficos/tabela acima, é que caso se atenda aos *datasets* mais pequenos (100 e 1000), não se observa grande diferença entre os tempos de execução, algo que provavelmente aconteceria com outros *datasets* que rondassem os mesmos tamanhos.

Contudo, através desta pequena análise pode-se concluir que a abordagem por nós implementada precisaria de alguns ajustes no que diz respeito ao algoritmo de interseção de dados e ao algoritmo de recuperação dos *hashes*, de modo a tornar possível a sua utilização em *data mining*. Provavelmente uma boa opção de otimização, por exemplo, seria a aplicação de um algoritmo de pesquisa binária.

### Dados trocados entre entidades

Outro tópico importante de avaliar neste tipo de protocolos é a quantidade de dados trocados entre as diferentes entidades presentes durante a comunicação e interseção dos mesmos. Este ponto permitirá também verificar a eficiência e escalabilidade de cada abordagem, assim como efetuar um balanço entre estes 2 aspetos e a privacidade/segurança oferecida por estas estratégias. Será de esperar, naturalmente, que quanto menos operações/trocas de dados existirem, menor será o valor calculado neste capítulo.

Novamente, voltam a ser usadas as 5 divisões do *dataset* original, de modo a ser possível efetuar uma análise mais completa de cada abordagem. São, de seguida, apresentados os resultados obtidos:





	100	1000	10000	100000	1000000
0	1440	16040	180040	2000040	20000040
2	10640	106040	1060040	10600040	106000040
3	53958	141394	1075522	10719570	106827282
Our	20802	210114	2308322	21928498	196232282

De novo, volta a ser possível observar uma tendência linear em qualquer um dos gráficos, significando que à medida que o número de registos do *dataset* aumenta, a quantidade de memória despendida também é maior. Este comportamento é algo de esperar (tal como no tempo de execução) devido à natureza das abordagens utilizadas.

Observando mais atentamente cada gráfico, a implementação que integra o *server trusted party* revela ser, novamente, a pior em termos de troca de dados. A principal razão para isto ter ocorrido deve-se, principalmente, à troca de mensagens encriptadas com *hash sha256* na íntegra, onde o número de *bytes* utilizados é 64. A abordagem com *Naive Hashing* demonstra ser, novamente, a menos custosa devido à forma como é desenvolvida, onde só são trocados os *hashes* de ambas as partes. Atendendo aos declives das retas de todos os gráficos é também possível observar diferenças entre as diversas estratégias. Neste caso, verifica-se que tanto a abordagem *Diffie-Hellman* como a *OT* apresentam um declive 5 vezes maior que a *Naive Hashing*, enquanto que a nossa implementação apresenta um declive 10 vezes maior. Ao analisar os dados trocados em cada tipo de *dataset* e nas diferentes abordagens, observa-se que inicialmente esta não é a tendência, mas, posteriormente, à medida que os *datasets* aumentam, estes declives começam a fazer mais sentido no contexto em causa.

Um resultado interessante de apontar será a similaridade dos valores apresentados pela abordagem *Diffie-Hellman* e *OT*. Exceto para os *datasets* originais (tamanho 100 e 1000), a quantidade de dados trocados acaba por ser bastante semelhante, manifestando valores ligeiramente maiores na abordagem de *OT*.

Como observação final, reitera-se a conclusão de que a implementação efetuada por nós ainda necessita de grandes ajustes, para que venha a tornar-se suficientemente eficiente e viável como as restantes abordagens. Uma possível melhoria ao nível deste aspeto poderia ser a substituição do algoritmo de *hashing* utilizado ou, em vez de enviar o *hash* completo, enviar apenas uma parte dele (embora isso possa resultar em perda de confiabilidade na interseção obtida posteriormente).

### Registos intersetados entre entidades

Algo bastante importante de atender durante a avaliação destas abordagens será a quantidade de dados intersetados, durante este processo de troca de dados, de modo a perceber o quão fiéis são. Relembra-se que cada método apresenta etapas bastante diferentes e que influenciam os resultados apresentados. Na seguinte tabela encontra-se apresentado as diferentes abordagens para os diferentes *datasets*.

	100	1000	10000	100000	1000000
0	52	544	6973	61240	650257
2	52	544	6972	61239	650238
3	52	544	6972	61237	650082
Our	52	544	6972	61238	650169
Original	52	544	6972	61238	650169

A tabela acima permite uma compreensão mais clara dos resultados apresentados em relação ao tempo de execução e à troca de dados. É possível ainda observar a existência de um *tradeoff* entre eficiência (tempo e memória) e fidelidade dos dados intersetados, onde quanto maior a eficiência, menos fidedignos os dados obtidos são e vice-versa.

Ao nível deste ponto, as abordagens que utilizam *Naive Hashing* e *Diffie-Hellman*, a partir de *datasets* de 10000 registos, já começam a apresentar alguns problemas, sendo detetadas mais interseções do que é suposto. Isto acontece devido à falta de precisão que estas estratégias manifestam, visto frequentemente serem apresentados *hashes* com apenas os 14/15/16 *bytes* mais significativos, não havendo um valor constante para o *output*. Já a abordagem *OT*, curiosamente, apresenta um comportamento contrário a partir de *datasets* com tamanho igual ou superior a 100000, observando-se que o número de interseções encontradas é relativamente menor. Uma possível razão para tal acontecer será o facto de não se ter conseguido “aprender” na totalidade os registos de ambas as partes, de modo a que fosse possível efetuar a interseção de conjuntos. Por fim, ao analisar mais detalhadamente a implementação criada, torna-se possível entender melhor as razões por trás de sua ineficiência. Esta utiliza algoritmos que são 100% fiáveis, não existindo cortes no que toca à informação trocada entre os dois lados da comunicação.

Como referidos nos 2 capítulos anteriores, seria necessário uma reformulação na abordagem implementada, de modo a aumentar a sua eficiência. Mas complementando com o que foi visto neste capítulo, seria igualmente interessante manter a fidelidade na interseção entre 2 *datasets*.

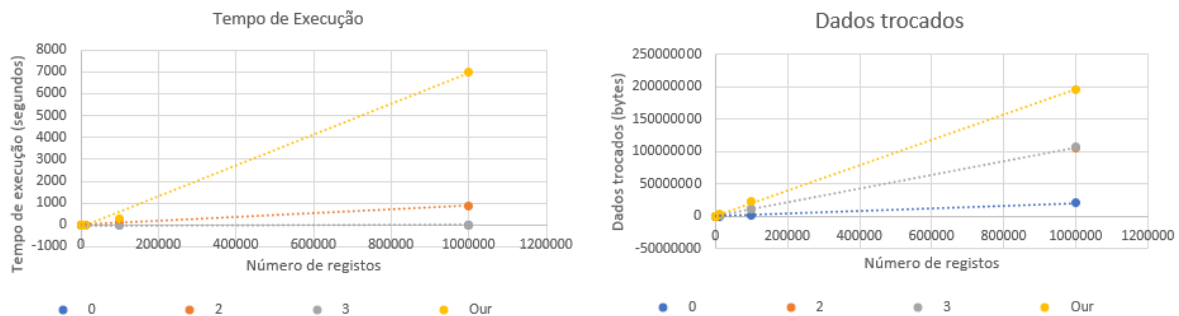
### Nível de segurança e privacidade

Por fim, são consideradas as características mais importantes que estes protocolos devem seguir nas soluções apresentadas: segurança e privacidade. Este constituiu um ponto essencial, no qual é necessário estabelecer um equilíbrio com as métricas previamente calculadas, a fim de compreender quais algoritmos realmente apresentam um bom desempenho.

- Naive Hashing: fornece alguma limitação quanto à privacidade, devido a ser só utilizado *hashes* como meio de intersetar conjuntos de dados, além de que, através de *dictionary attacks*, atacantes podem potencialmente inferir quais são os registos originais. Para que ambas as partes tenham privacidade é ainda necessário assegurar que os registos trocados apresentem uma certa entropia entre si. A nível da segurança, esta abordagem poderá revelar ser vulnerável a *collision attacks*, podendo implicar alguns erros na conversão para *hash* de certos registos.
- Diffie-Hellman: a nível de privacidade esta abordagem fornece resultados bastante positivos, visto basear-se num algoritmo assimétrico robusto. Já a nível de segurança, esta estratégia apoia-se no problema do logaritmo discreto e, como tal, este deve ser bem definido de modo a que não seja facilmente resolvido. Além disto, os mecanismos de autenticação devem ser também suficientemente robustos, caso contrário, torna-se fácil efetuar *man-in-the-middle attacks*.
- OT-based: através desta abordagem é assegurada privacidade em ambas as partes, já que cada utilizador só “aprende” certos registos de outro, nunca sendo possível conhecer quais são os registos de ambas as partes na sua totalidade, apenas a interseção destas. Na segurança são seguidos fortes princípios matemáticos e, como tal, estes devem encontrar-se bem definidos, para que este esquema não seja facilmente quebrado.

- Implementação Server Trusted Party: o nível de segurança e privacidade, neste caso, irá depender em grande parte do servidor responsável por partilhar, com os dois lados da comunicação, a interseção dos dados. É necessário que este ponto intermediário seja o mais seguro possível, de modo a que nenhuma das partes seja “enganada”. Outro aspecto negativo é a presença de um *single point of failure*, o que torna esta abordagem facilmente vulnerável a ataques.

De modo a ter melhor percepção das comparações efetuadas anteriormente relativamente ao tempo de execução e quantidade de dados trocados pelos diferentes protocolos, são apresentados os seguintes gráficos:



## Sistema de recomendação

O problema original consistia em encontrar a interseção dos *ratings* de um utilizador em duas plataformas de *streaming* de música, de modo a que as mesmas conseguissem efetuar sistemas de recomendação mais precisos. Visto que a implementação de um sistema de recomendação não é algo explicitamente solicitado no enunciado, será explicada a ideia para a sua implementação.

Antes de falar deste tipo de sistemas é necessário entender que existem 2 tipos de implementações principais: *content filtering* e *collaborative filtering*. O primeiro visa efetuar um sistema em que sejam recomendados *items* a um determinado utilizador consoante as características destes. Por exemplo, se determinado indivíduo vê bastantes filmes de ação com determinado ator, o sistema de recomendação sugerirá filmes que apresentem características bastante semelhantes a estes. A segunda abordagem pretende efetuar um sistema em que sejam recomendados *items* consoante as opiniões providenciadas pelos indivíduos em redor. Por exemplo, se num supermercado em uma determinada faixa etária for normal comprar pasta de dentes, todos os indivíduos desta faixa etária receberão a sugestão de comprar este mesmo produto.

O sistema de recomendação, neste caso, surgiria como um híbrido dos dois tipos de implementações existentes. Na prática, funcionaria da seguinte forma:

- duas plataformas (*Spotify* e *Soundcloud*) têm os *ratings* atribuídos por um determinado utilizador a certas músicas.
- as plataformas enviam os *ratings* para o servidor e o mesmo devolve a interseção dos mesmos para as respectivas plataformas.
- sabendo que um utilizador fez questão de expressar o seu *rating* relativamente a uma determinada música, o *Spotify* e o *Soundcloud* podem usar os *ratings* *intersectados* para se basearem e construírem sistemas de recomendação.
- estes sistemas de recomendação terão, assim, como foco apenas as músicas intersectadas, evitando quaisquer desvios que outros *ratings* possam causar.

- as recomendações efetuadas poderão tanto basear-se em, por exemplo, *ratings* que os amigos providenciaram, como nas características das músicas intersectadas.

Esta seria a ideia a ser seguida para implementar os sistemas de recomendação após ter sido efetuada a interseção de conjuntos de *ratings*, atribuídos por um utilizador, numa plataforma de *streaming*. Apesar de não ter sido implementada, acredita-se que seja uma ideia viável e com potencial. Reduzir o conjunto de dados em alguma percentagem, ao ser efetuada a sua interseção, acaba também por ser benéfico, já que permitirá efetuar recomendações mais precisas para um determinado utilizador.

## Conclusão

Através deste trabalho conseguiu-se compreender melhor a importância e o funcionamento de protocolos *PSI* na troca de informação entre duas entidades. Estes apresentam um papel fundamental na manutenção da privacidade e segurança dos dados durante a interseção dos mesmos. Através destes protocolos, foi-nos ainda solicitado a resolução de um problema.

Inicialmente, começou-se por definir o *dataset* a utilizar, neste caso, um conjunto de músicas e os respetivos *ratings* atribuídos por um utilizador. De seguida, procurou-se compreender a melhor abordagem para explorá-lo, a fim de realizar divisões no conjunto de dados que permitissem testar os diferentes protocolos da maneira mais eficaz. Chegou-se à conclusão que a utilização de tamanhos 100, 1000, 10000, 100000 e 1000000 seriam os mais indicados para avaliar as diferentes abordagens aplicadas aos protocolos.

Começou-se por ter um primeiro contacto com as ferramentas e a implementação por nós efetuada para a abordagem *server trusted party*, de modo a entender o comportamento dos diversos protocolos. Para este processo foi utilizado um *dataset* base com 2066 músicas e *ratings*. Posteriormente, efetuou-se uma análise mais minuciosa, procedendo-se à realização do respetivo *benchmark*, de modo a avaliar métricas como tempo de execução, quantidade de dados transmitida e quantidade de interseções encontradas. Através deste processo, conseguiu-se entender que existe um grande *tradeoff* entre eficiência e nível de privacidade/segurança, já que os mesmos são inversamente proporcionais.

Finalmente, equacionou-se a possibilidade de, utilizando estes protocolos, ser resolvido um problema por nós definido, neste caso, sistemas de recomendações com base nos *ratings* de músicas atribuídos por um determinado utilizador. Conclui-se, assim, que este seria um objetivo viável e atingível utilizando os protocolos *PSI*.

## Referências

- Slides da disciplina de Segurança e Privacidade
- <https://eprint.iacr.org/2008/197.pdf>
- <https://eprint.iacr.org/2015/634.pdf>
- <https://www.geeksforgeeks.org/socket-programming-cc/>
- <http://www.zedwood.com/article/cpp-sha256-function>
- <https://github.com/rscmendes/PSI>
- [https://www.kaggle.com/code/manuelbenedicto/eda-on-spotify-global-2019-most-streamed-tracks/input?select=spotify\\_global\\_2019\\_most\\_streamed\\_tracks\\_audio\\_features.csv](https://www.kaggle.com/code/manuelbenedicto/eda-on-spotify-global-2019-most-streamed-tracks/input?select=spotify_global_2019_most_streamed_tracks_audio_features.csv)
- <https://www.kaggle.com/code/keremkarayaz/spotify-song-list/input>